ISSN:0975-9646

Improving the Performance of Processing for Small Files in Hadoop: A Case Study of Weather Data Analytics

Guru Prasad M S[1], Nagesh H R [2,] Deepthi M [3]

Dept of CSE [1,] Dept of CSE [2], Dept of CSE [3]
SDMIT [1], MITE [2], JNNCE [3]
Ujire, India [1], Moodabidri, India [2], Shimoga, India [3]

Abstract--Hadoop is an open source Apache project that supports master slave architecture, which involves one master node and thousands of slave nodes. Master node acts as the name node, which stores all the metadata of files and slave nodes acts as the data nodes, which stores all the application data. Hadoop is designed to process large data sets (petabytes). It becomes a bottleneck, when handling massive small files because the name node utilize more memory to store the metadata of files and the data nodes consumes more CPU time to process massive small files. In this paper, the author proposes the Optimized Hadoop, consists of Merge Model to merge massive small files into a single large file and introduced the efficient indexing mechanism. Our experimental result shows that Optimized Hadoop improves performance of processing small files drastically up to 90.83% and effectively reduces the memory utilization of the name node to store the metadata of files.

Keywords—Hadoop; Hadoop Distributed File System,; Map Reduce; Small Files.

I. INTRODUCTION

In recent years, Hadoop has become a most popular high performance distributed computing paradigm for large scale data analytics [1]. The Hadoop architecture consists of the Hadoop Distributed File System (HDFS) and a MapReduce programming model. HDFS is high fault tolerance, high throughput, and high reliability, designed to deploy on commodity hardware. MapReduce is a programming model proposed by Google [2], to process large data sets.

Hadoop is excellent in handling large files of data; HDFS divides the input data into data blocks of size 64 MB. NameNode stores the metadata of the data blocks and DataNodes stores the data blocks. These data blocks are processed by the MapReduce.

Hadoop is inefficient in handling massive small files, whose size ranges from 10KB to 5 MB. Massive small files are generated by weather sensors, word docs, power point, flash files, images of maps, MP3, video clips and so on [5]. These kinds of files will bring serious problems to Hadoop performance. First, storing too many small files into Hadoop becomes overhead in terms of memory usage of metadata stored in the NameNode; this will impact on the size of the memory in the NameNode. Secondly, more number of MapReduce task created to process massive small files and it creates overhead between MapReduce tasks and CPU time.

To overcome these problems, the author proposes the Optimized Hadoop consists of Merge model. It merges all the input files into a single large file and this single large file moving into HDFS. HDFS divides the single large file into data blocks of size 64 MB. NameNode stores metadata of files and DataNode store data blocks. The Optimized Hadoop reduces memory usage by the NameNode to store metadata, reduces overhead created between MapReduce tasks and improves the performance of DataNodes to process data blocks.

The major contributions of this paper are summarized as follows:

- Effective number of MapReduce task created to process HDFS data blocks, this drastically reduces MapReduce task overhead and the total CPU time.
- Efficient metadata management will successfully reduce the memory utilization of the NameNode to store metadata files.
- Optimized Hadoop is not just suitable for weather data files; it can be applied universally to all types of small files.

The rest of this paper is organized as follows. Section II describes background of the Hadoop Distributed File System and the MapReduce. Section III explores the small files problems. Section IV provides the proposed model. Section V presents performance evaluation and discussion. Conclusion and future work are drawn in Section VI.

II. BACKGROUND

A. *Hadoop Distributed File System*

Hadoop two fundamental subprojects are the HDFS and the MapReduce. The distributed file system named by Hadoop Distributed File System (HDFS) is a designed to run on commodity hardware [3]. The block size of HDFS is much larger than that of normal file system i.e. 64MB by default. The reason for the large size of blocks is to reduce the number of disk seeks. This is not a POSIX-compliant file system, and once data is written to file system it can't be modified (a write-once, read-many access model). HDFS protects data by replicating data blocks into multiple nodes, with a default replication factor of 3.One major usage of HDFS is which has very good durability

HDFS has a master/slave architecture which consists of two important agents, NameNode and DataNode. Figure 1 shows the Hadoop Distributed File System. The master, called the NameNode which is responsible for managing file system namespace, maintains

the file system tree and all metadata and file system actions within the HDFS (e.g. Files list and files'sub-blocks location information) and there are number of slaves, called the DataNodes which are responsible for actual data I/O. The DataNodes service all read/write and file replication requests based on direction from the NameNode. Because Hadoop keeps all file system metadata in main memory, it is necessary for the NameNode to be have own server, this way file access is not slowed because of strain on the NameNode from serving metadata requests. Without the NameNode it is not possible to access the file. So it becomes very important to make NameNode resilient to failure

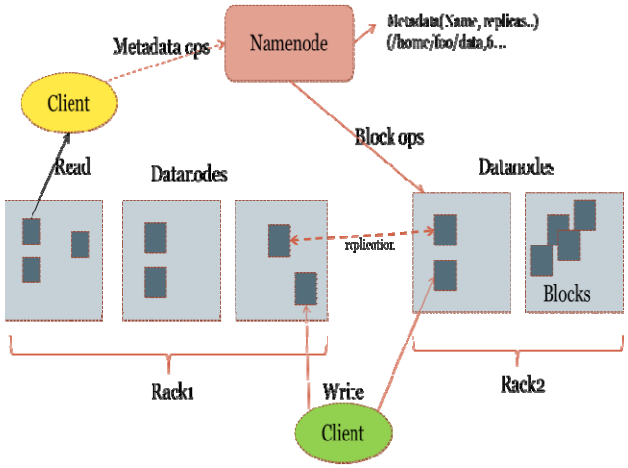

Fig.1. Hadoop Distributed File System.

B. *MapReduce*

MapReduce is a programming model from Google for the purpose of supporting its critical services such as web search, log analysis, data mining, etc [2]. This model designed to efficiently execute programs on large clusters, by exploiting data parallelism and comprises of Map phase and Reduce phase. In Map phase mapper must be able to ingest the input and process the input record and that processed record will be forwarded to Reduce Phase, there task will reduced.

The Map function takes in a key/value pair and outputs an intermediate list of key/value pairs i.e. Map (k1, v1) → list (K2, v2). The Reduce functions will then take all values associated to the same key and produce the final output list of key/values i.e. Reduce (K2, list (v2)) → list (v3). . The map creates several output files, those records are sorted by key. One of the important advantages of the above schema is that the parallelization complexity is handled. But this advantage often leads to loss of flexibility.

Every job must consist of exactly one Map function and followed by an optional Reduce function, these steps cannot be executed in a different order. And also if an algorithm requires multiple Map and Reduce steps that can be enforced by separate jobs, and data can only be transferred from one job to the next, through the file system (HDFS).

In the initial implementations of Hadoop, Map Reduce is designed as a master-slave architecture .which incorporated by JobTracker and TaskTrackers. The JobTracker is the master which carries off the cluster resources, scheduling jobs, monitoring progress and dealing with fault-tolerance along with that it will distribute the tasks and their input split to the various trackers. On each of the slave nodes, there exists a TaskTracker which is responsible for launching parallel tasks and reporting their status to the JobTracker. The TaskTracker service will actually run our map and reduce tasks,

III. THE SMALL FILES PROBLEM

This section explores the impact of small files on the Hadoop.

A. *Impact on time taken to move files into HDFS*

Before running the Hadoop jobs, input files are copying from local file system into Hadoop Distributed File System. Larger numbers of small files will take more time to copy from local file system into Hadoop Distributed File System.

B. *Impact on memory usage of the NameNode*

Hadoop is a Master/Slave architecture consists of one Master (NameNode) and many slaves (DataNodes). Hadoop Distributed File System divides the input data into data blocks. NameNode stores the metadata of each block and DataNodes stores the data blocks. Each metadata consumes about 150 bytes of the NameNode memory [8]. For larger number of small files more numbers of metadata created and it consumes more memory of the NameNode.

C. *Impact on time taken to process files*

HDFS divides the larger input file into data blocks of size 64 MB (i.e. by default) and these data blocks were processed by the MapReduce. Small files, whose size less than 64 MB will occupy one data block each and more number of MapReduce tasks created to process massive data blocks. It creates overhead between MapReduce tasks and more time taken to process files.

IV. PROPOSED MODEL

The proposed model extends Hadoop and has been named as Optimized Hadoop. The basic idea of our proposed Optimized Hadoop consists of Merge Model. The Merge Model algorithm is as follows:

1. Initially returns the array of abstract path names defining the files in the directory as an input directory.
2. Loop for i=0; i<files.length; increment i
   a) Get the actual path of files and read them
   b) Insert the lines into the output file.
   c) Rea the file till end of file while (line!=null)
3. Display the message that files is merged or if any error exception is shown.

In the Optimized Hadoop, Merge Model combines massive small files into a single large file. This large file moved into HDFS. HDFS divides a large file into data blocks of size 64 MB (i.e. by default). Each data blocks are processed by the MapReduce.

The Optimized Hadoop solves the small files problems as follows:

1. Reduces time to move file from local file system to Hadoop Distributed File System.
2. Minimizes the memory usage by the NameNode to store metadata of files.
3. Improves the performance of processing for small files.

V. PERFORMANCE EVALUATION

The Performance of the Hadoop cluster with respect to the time taken to store files into the Hadoop Distributed File System , memory usage of the NameNode and time taken to process files was initially benchmarked with the original Hadoop (traditional) and then compared with results obtained using the Optimize Hadoop (proposed).

A. Experiment Environment

The experimental environment is built on a cluster with five machines. One machine acts as the NameNode and the other four machine acts as the DataNodes. Each of these machines has Intel® Xeon® E5520@2.27 GHz processor, 2 GB RAM, and 500 GB SATA hard disk and operating system is Ubuntu 10.0.4. Hadoop version is 1.0.3 and the java version is 1.6.0. The number of replicas is set to 2 and the HDFS block size is 64MB

B. Workload Overview

The workload consists of 1,018 weather data files which is of size 2 GB; they are generated by weather sensors located across the globe [4]. The File size ranges from 250 KB to 5000 KB. Figure 2 shows the distribution of file size.

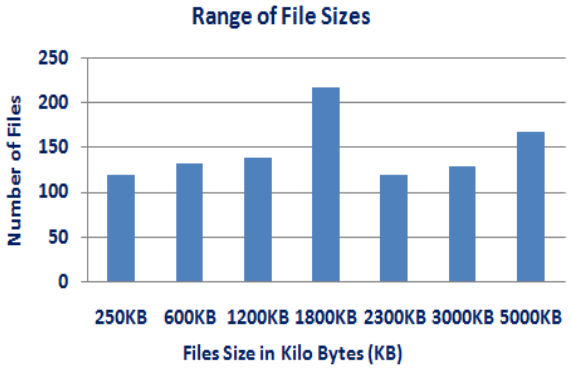

Fig.2. Size distribution of files

C. Performance Measurement Parameters

The performance of the Hadoop cluster was measured on the following parameters.

1. Time taken to move files from local file system to HDFS
2. Memory usage of the NameNode to store metadata.
3. Time taken in the MapReduce phase to process files.

D. Time taken to move files in to the HDFS

Files moving operation was performed on both the original Hadoop and optimized Hadoop. There we recorded the time taken to store files into the Hadoop Distributed File System. Table I shows the time taken S by the original Hadoop (traditional) and the optimized Hadoop (proposed) to move files into HDF. Figure 3 shows the chart of the time taken by the original Hadoop (traditional) and the optimized Hadoop (proposed) to move files in to HDFS.

Table I. Time taken to move files in to the Hadoop Distributed File System

Technique	File Size in GB	Time Taken in seconds
Original Hadoop (traditional)	02	162
Optimized Hadoop (proposed)	02	71

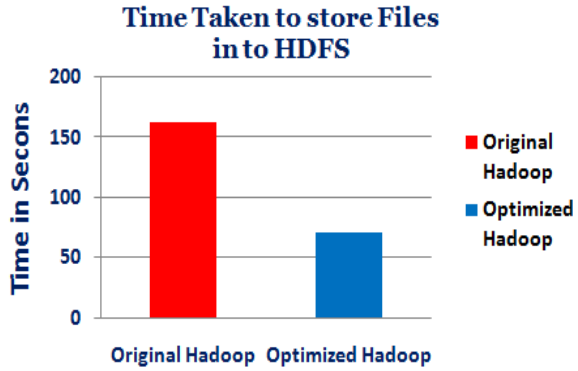

Fig.3. Time taken to move files in to Hadoop Distributed File System

E. Measurement of memory Usage of the NameNode

Hadoop Distributed File System divides the input data into data blocks of size 64 MB (i.e. by default). It stores the metadata of each block in the NameNode (Master Node) and all the data blocks in the DataNodes (Slave Nodes). In the NameNode, metadata of each block consumes around 150 bytes of memory.

In the case study of Weather data contains 1018 small files; total size of these files is 2 GB. In the original Hadoop, the HDFS created 1018 data blocks, because the input data contains 1018 small files whose size less than 64 MB. Memory usage of the NameNode to store 1018 metadata of data blocks was 152700 bytes. In the optimized Hadoop consists of the Merge module, it combines 1018 small files into a single large file of size 2 GB and moved this into the HDFS. HDFS divides 2 GB file into 32 data blocks. Memory usage of the NameNode to store 32 metadata of data blocks was 4800 bytes.

Table II shows the memory usage of the NameNode in the original Hadoop (traditional) and in the optimized Hadoop (proposed). Figure 4 shows the chart of the memory usage of NameNode in the original Hadoop (traditional) and in the optimized Hadoop (proposed).

Table II. Memory usage of the NameNode

Technique	Memory usage in bytes
Original Hadoop (traditional)	152700
Optimized Hadoop (proposed)	4,800

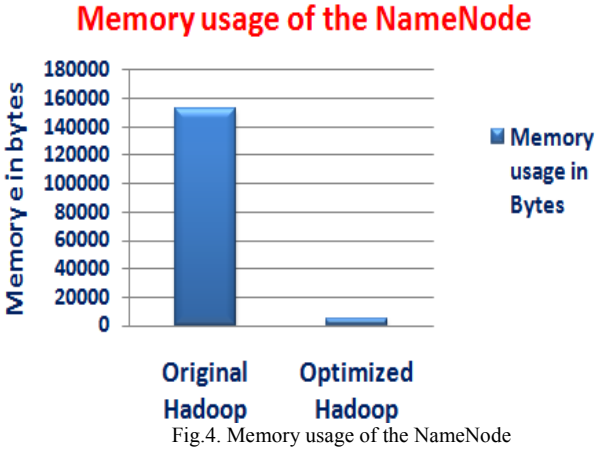

Fig.4. Memory usage of the NameNode

F. Time taken to process files.

Hadoop is designed to process large files. The task of processing large numbers of files of smaller size degrades the performance of Hadoop system. The case study of Weather data analytics contains 1018 small files [8]; total size of these files is 2 GB. When all these files are processed individually, the Map phase takes a time of 3998 seconds and the Reduce phase takes a time of 104 seconds. Hence total CPU time taken is 4102 seconds. In our optimized Hadoop, the Merge module merged these 1018 files into a single large file of size 2 GB. To process this single large file, the time taken by the Map phase is 340 seconds and by the Reduce phase is 36 seconds, hence the total CPU time taken is 376 seconds. The proposed optimized Hadoop improves the performance 91.49%, 65.38% and 90.83 % of Map time, Reduce time and total CPU time respectively.

Table-III shows the comparison of Map time, Reduce time and total CPU time of the original Hadoop (Conventional) and the optimized Hadoop (Proposed) of Weather data analytics. Figure 5 describes the chart of Map time, Reduce time and total CPU time of the original Hadoop (Conventional) and the optimized Hadoop (Proposed) of Weather data analytics

Table-III .Comparison of Map, Reduce and Total CPU time of the original Hadoop (traditional) and the optimized Hadoop (proposed)

Technique	File Size In GB	Map Time in seconds	Reduce Time in Seconds	Total CPU Time in Seconds
Original Hadoop (traditional)	02	3998	104	4102
Optimized Hadoop (proposed)	02	340	36	376

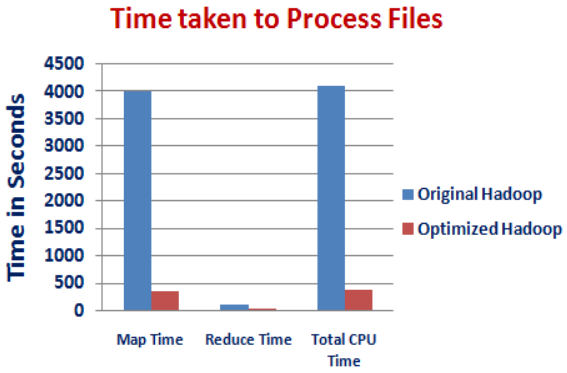

Fig.5.Chart of Map, Reduce and Total CPU time of the original Hadoop (traditional) and the optimized Hadoop (proposed)

VI CONCLUSION AND FUTURE WORK

Hadoop is deployed to process large files and suffers a performance penalty while processing a large number of small files. In addition, the memory usage of the NameNode to store metadata increases rapidly and more time taken to move files from local file system to Hadoop Distributed File System. The proposed Optimize Hadoop minimizes the memory usage of the NameNode to store the metadata of files, reduces the time taken to move files from local file system to Hadoop Distributed File System, efficiently managed small files and enhance the performance of processing inherently small input files. The experiment results show that our Optimized Hadoop effectively improves the efficiency of storing, managing and processing small files. Our strong findings are as follows: (1) Time is taken to move files from local file system to Hadoop Distributed File System is reduced from 162 seconds to 71 seconds. (2) Memory usage by the NameNode to store metadata has decreased from 1, 52,700 bytes to 4,800 bytes. (3) Improves performance of processing small files drastically up to 90.83%.

Our further work will include finding other parameters that impact on the Hadoop performance.

REFERENCES

[1] Apache Software Foundation. Official apache Hadoop website, http://hadoop.apache.org/, Aug, 2012.

[2] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters" in Google, *Inc.*

[3] The Hadoop Architecture and Design, Available: http://hadoop.apache.org/common/docs/r0.16.4/hdfs_design. html, Aug, 2012.

[4] National Climatic Data Center (NCDC) http://www.ncdc.noaa.gov/

[5] Bo Dong, Jie Qiu and Qinghua Zheng "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files" IEEE International Conference on Services Computing, 978-0-7695-4126-6/10, 2010.

[6] Hung-Chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker from Yahoo and UCLA, "Map-Reduce- Merge: Simplified Data Processing on Large Clusters", paper published in Proc. of ACM SIGMOD, pp. 1029– 1040, 2007.

[7] K. Schvachko, H. Kuang, S. Radia, R. Chansler. "The Hadoop Distributed File System". In Proceedings of IEEE 26th symposium on Mass Storage Systems and Technologies (MSST), Incline Village, Nevada, USA, May 2010.

[8] Tom White, "The Small Files Problem". http://www.cloudera.com/blog/2009/02/the small files problem, 2009.

[9] Chuck Lam, Hadoop In Action, 1st ed. Dec. 2010, pp. 8.

[10] Tom White, Hadoop: The Definitive Guide, 2nd ed. O'Reilly Media Yahoo Press, Jun. 2009, pp. 41 45.